Санкт-Петербургский государственный университет

Кафедра информатики

Романов Артем Витальевич

Алгебраические байесовские сети: система анализа и синтеза вторичной структуры

Бакалаврская работа

Научный руководитель:
проф. каф. инф., д .ф.-м. н., доц. Тулупьев А. Л.

Рецензент:
доцент НИУ ВШЭ СПб., к. ф.-м. н. Сироткин А. В.

Санкт-Петербург
2016

SAINT-PETERSBURG STATE UNIVERSITY

Computer Science Department

Romanov Artem Vitalyevich

Agebraic Bayesian networks: system for analysis and synthesis of secondary structure

Bachelor's Thesis

Scientific supervisor:
doctor. of Sc., associate professor Tulupyev Alexander

Reviewer:
associate professor NRU HSE, PhD Sirotkin Alexander

Saint-Petersburg
2016

Оглавление

Введение

Алгебраические байесовские сети (АБС), представляющие собой разновидность графических вероятностных моделей [29, 1], на данный момент активно исследуются с точки зрения автоматического синтеза глобальных структур таких моделей [26].

Среди глобальных структур АБС получили формальное определение и рассматриваются первичная, вторичная, третичная и четвертичная. Проблема синтеза вторичной структуры, представленной в виде минимального графа смежности, по первичной, рассматривается в [26].

Несмотря на то, что в данный момент для операций с алгебраической байесовской сетью используют ее вторичную глобальную структуру, третичная также применяется как для генерации случайного минимального графа смежности, так и для порождения всего множества минимальных графов смежности [20].

В данной работе рассматривается проблема инкрементализации синтеза третичной и четвертичной структур, а именно, исследуется возможность перестроить данные структуры при добавлении новой вершины в первичную, вместо того, чтобы их синтезировать по всему получившемуся набору заново.

Настоящая выпускная квалификационная работа бакалавра является проектной, то есть выполняется совместно с коллегами (Левенцом Даниилом Григорьевичем, Березиным Алексеем Ивановичем), кроме того, тесно связана с ВКР Мальчевской Екатерины Андреевны. Проект представляет собой комплекс программ по синтезу, анализу и визуализации вторичной структуры АБС на платформе .NET с использованием языка C# и свободно распространяемой библиотеки GraphSharp.

Комплекс программ позволяет синтезировать все глобальные структуры алгебраической байесовской сети по набору вершин, добавлять и удалять вершины, изменяя при этом соответствующим образом все структуры, а также генерировать все множество минимальных графов смежности.

Эта работа является частью более широких инициативных проектов, выполняющихся в лаборатории теоретических и междисциплинарных основ информатики СПИИРАН под руководством А.Л. Тулупьева; кроме того, разработки были частично поддержаны грантами РФФИ 15-01-09001-а — «Комбинированный логико-вероятностный графический подход к представлению и обработке систем знаний с неопределенностью: алгебраические байесовские сети и родственные модели», 12-01-00945-а — «Развитие теории алгебраических байесовских сетей и родственных им логико-вероятностных графических моделей систем знаний с неопределенностью», 09-01-00861-а —«Методология построения интеллектуальных систем поддержки принятия решений на основе баз фрагментов знаний с вероятностной неопределенностью», а также проектом по ФЦНТП «Исследования и разработки по приоритетным направлениям развития науки и техники на 2002–2006 годы», 2006-РИ-19.0/001/211, Государственный контракт от «28» февраля 2006 г., № 02.442.11.7289, «Направленные циклы в байесовских сетях доверия: вероятностная семантика и алгоритмы логико-вероятностного вывода для программных комплексов с байесовской интеллектуальной компонентой».

1. Обзорная часть

1.1. Введение

В данной главе проводится обзор теории алгебраический байесовских сетей, касающейся синтеза глобальных структур. Обосновывается актуальность выбранной для работы темы. Также формализуется цель работы — инкрементализация синтеза третичной и четвертичной структур — и задачи, выполение которых необходимо для ее достижения. Приводятся преимущества выбранных средств разработки, а именно языка C# и Microsoft Visual Studio 2015.

1.2. Актуальность работы

Алгебраическая байесовская сеть представляет собой разновидность графических логико-вероятностных моделей систем знаний с неопределенностью [3]. Исследования на тему АБС направленны на решение двух актуальных проблем: дефицита знаний и математических моделей для представления таких знаний с неопределенностью [8, 9].

Выгодное отличие АБС заключается в том, что она позволяет работать не только со скалярными оценками представления неопределенности в знаниях, как это происходит в Байесовских сетях доверия, но и с интервальными [10, 8]. Кроме того, алгебраические байесовские сети позволяют проводить такие операции как поддержание непротиворечивости сети, осуществление априорного и апостериорного логико-вероятностного вывода [7, 11, 13].

В теории АБС особое внимание уделяют первичной и вторичной структурам. Первичная представляет собой множество фрагментов знаний, характеризующих некую предметную область, а вторичная — определяет взаимосвязи между ними. Вторичную структуру представляют минимальным графом смежности, что обсуловлено тем, что если над заданной первичной структурой возможно построение ациклических графов смежности, то их множество будет совпадать с множеством минимальных графов смежности [19]. Ацикличность первичной струк-

туры, в свою очередь, обусловленна тем, что они являются на данный момент одним из условий работы алгоритмов поддержания непротиворечивости и глобального априорного вывода [11, 15, 16].

Проблема синтеза вторичной, а так же третичной и четвертичной структур была рассмотрена в работах [26, 20, 21].

Среди глобальных структур алгебраической байесовской сети также выделяют третичную структуру, инкрементальные алгоритмы синтеза которой представленны в данной работе. Изначально, третичную структуру использовали как вспомогательную для синтеза вторичной [5, 17]. В дальнейших исследованиях область ее применения расширилась до выявления ацикличности [19], построения всего множества минимальных графов смежности [22, 23, 24], а также синтеза случайного минимального графа смежности и поиска наиболее эффективной вторичной структуры [20]. Кроме того, в некоторых работах вопрос о возможности осуществления всего глобального логико-вероятностного вывода на третичной структуре, без использования вторичной, признается авторами актуальным [25].

Одним из насущных вопросов в данной области является разработка алгоритмов обучения сети [12, 14, 9], в частности, автоматического синтеза глобальных структур. Алгоритмы синтеза вторичной, третичной и четвертичной структур рассмотрены в работах [5, 22, 23, 24, 26]. Но если при изменении множества фрагментов знаний алгебраической байесовской сети использовать разработанные алгоритмы, придется заново перестраивать все глобальные структуры, что влечет за собой значительные временные затраты. Поэтому, исследования, в ходе которых были разработаны инкрементальные алгоритмы синтеза глобальных структур, в частности вторичной, опубликованные в [2, 6], и третичной, разработанные автором и представленные в данной работе, затрагивают актуальную и нерешенную проблему.

Последняя из еще не рассмотренных выше глобальных структур — четвертичная — так же, как и третичная, используется для синтеза всего множества минимальных графов смежности [26, 18, 21]. Разработанные автором инкрементальные алгоритмы синтеза четвертичной

структуры при удалении или добавлении вершины в первичную структуру, представленны в 3-й главе.

Ввиду того, что в данной работе исследования проводятся в контексте глобальных структур алгебраических байесовских сетей, не затрагивая аппарат логико-вероятностного вывода, в том числе такой его элемент как фрагмент знаний, будем оперировать понятием вершины, а именно, добавлением и удалением вершин из первичной структуры АБС. Вершины однозначно задаются своей нагрузкой — подмножеством некоторого заданного алфавита, причем нагрузки попарно не совпадают и не поглощают одна другую.

1.3. Цели и задачи

Целью данной работы являлась инкрементализация синтеза третичной и четвертичной структур алгебраических байесовских сетей.

Для реализации цели были поставлены и выполнены следующие задачи.

Теоретические представлены ниже.

1. Разработать инкрементальный алгоритм синтеза третичной структуры, обеспечивающий синтез третичной структуры, используя уже построенную, при добавлении либо удалении вершины в первичную структуру.

2. Разработать инкрементальный алгоритм синтеза четвертичной структуры, обеспечивающий синтез третичной структуры, используя уже построенную, при добавлении либо удалении вершины в первичную структуру.

Технологические представлены ниже.

1. Осуществить программную реализацию разработанных алгоритмов в виде библиотеки.

2. Интегрировать библиотеку в совместно разрабатываемый комплекс программ по синтезу и анализу вторичной структуры алгеб-

раических байесовских сетей.

1.4. Средства разработки

Языком и платформой для разработки проекта был выбран C# и .NET framework, так как они обладают рядом свойств, значительно упрощающих разработку [27]:

1) атрибуты с декларативными метаданными о типах во время выполнения;

2) автоматическая сборка мусора;

3) поддержка xml–документации;

4) поддержка языка запросов LINQ.

В качестве среды разработки была выбрана Microsoft Visual Studio 2015, имеющая ряд преимуществ [28]:

1) динамический анализ кода;

2) удобное управление подключаемыми библиотеками с помощью диспетчера пакетов NuGet;

3) отсутствие необходимости перестраивать решение.

Для визуализации структур были использованы технологии WPF (Windows Presentation Foundation) и свободная библиотека GraphSharp, предоставляющая возможности для хранения графов и построения их диаграмм.

1.5. Выводы по главе

Была сформулирована цель работы, а также поставлены задачи, выполнение которых необходимо для ее достижения. Описаны выбранные средства разработки и их достоинства.

2. Определения и обозначения

2.1. Введение

В данной главе описываются понятия и определения из теории алгебраических байесовских сетей, используемые в дальнейшем в работе.

2.2. Используемые определения и понятия

Чтобы сформировать систему обозначений, приведем определения и понятия, уже сложившиеся в [4, 26].

***Определение* 2.1.** [4] *Графом $G = (V, E)$ называется совокупность двух множеств — непустого множества V (множества вершин) и множества E двухэлементных подмножеств множества V (E — множество ребер).*

***Определение* 2.2.** [26] *Нагруженный граф — тройка (G, A, W), где G — граф, A — алфавит атомов, W — функция нагрузки, заданная на множествах вершин и ребер G со значениями из 2^A.*

***Определение* 2.3.** [26] *Сепаратором двух вершин в нагруженном графе назовем пересечение нагрузок соответствующих вершин.*

В контексте данной работы будем для удобства отождествлять понятие вершины v и нагрузки вершины W_v, заменяя одно на другое. Также будем рассматривать только согласованные графы, то есть такие, у которых нагрузка любого ребра равна пересечению нагрузок вершин этого ребра.

***Определение* 2.4.** [26] *Максимальный граф смежности $G_{\max}$ — наибольший по числу ребер граф смежности.*

***Определение* 2.5.** [26] *Сужение максимального графа смежности G на сепаратор u — это подграф графа G, в который входят только те вершины и ребра, нагрузки которых содержат или равны u.*

***Определение* 2.6.** *Третичная структура — это граф, представленный в виде диаграммы Хассе с порядком следования сверху вниз, построенный на множестве непустых сепараторов.*

***Определение* 2.7.** *Родителем сепаратора будем называть сепаратор, предшедствующий данному в третичной структуре.*

***Определение* 2.8.** *Сыном сепаратора будем называть сепаратор, следующий за данным в третичной структуре.*

***Определение* 2.9.** [26] *Полусиблинговый граф сепаратора u — это граф, построенный над множеством его сыновей, ребро между двумя вершинами которого проведено тогда и только тогда, когда сужения, соответствующие данным вершинам, пересекаются.*

Другими словами, сужения максимального графа смежности будут пересекаться только в том случае, если пересекаются множества вершин данных сужений. Такое возможно, только если более одной вершины принадлежит обоим сужениям, а тогда такие вершины содержат в себе оба сепаратора, по которым строятся сужения.

Получается, что сужения двух сыновей сепаратора u пересекаются тогда, когда в максимальном графе смежности присутствует вершина, нагрузка которой содержит в себе оба сына.

***Определение* 2.10.** *Семейство полусиблинговых графов, построенных для каждого элемента множества сепараторов в объединении с пустой нагрузкой является четвертичной структурой.*

2.3. Используемые в листингах алгоритмов обозначения

Методы .Contains(element)(например VertexSeps.Contains(sep)) — применяется к множеству, возвращает true, если element содержится в множестве, false — в противном случае.

Обращение [key] к множеству (например VertexSeps[sep]) — применяется к структуре, представленной словарем. Возвращает значение из словаря по ключу key.

Обращение .Value к элементу (например sep.Value) — возвращает значение из элемента словаря, представленного парой ключ–значение.

Метод .Remove(element)(например Seps.Remove(sep)) — удаляет из множества элемент element, либо удаляет из словаря пару ключ-значение, ключ которой равен element.

Обращения .Source и .Target — применяются к элементам множества ребер направленного графа, возвращают соответственно вершину, из которой ребро проведено, и вершину, в которую оно проведено.

Обращение .V (например currentGraph.V) — возвращает множество вершин текущего графа.

Функция SonsFromThird(sep, G) — возвращает множество сыновей сепаратора sep из третичной структуры, представленной графом G.

Метод .AddVertex(v) — добавляет в множество вершин текущего графа вершину v.

Метод .AddEdge(s, v) — добавляет ребро, проведенное из вершины s в вершину v в множество ребер текущего графа.

Метод .AddNewGraph(s, NewGraph) — добавляет новую пару ключ-значение в словарь полусиблинговых графов, где s — это ключ, а так же сепаратор, по которому строится новый полусиблинговый граф. NewGraph — полусиблинговый граф, построенный по сепаратору s, являющийся значением в добавляемой паре.

Функция ConstrictionIntersect(s, v) — проверяет, пересекаются ли сужения сепараторов s и v. Возвращает true, если сужения пересекаются, false — в противном случае.

Метод .Add(sep, int) — добавляет в текущий словарь новую пару ключ–значение, где ключом является sep, а значением — второй параметр.

Метод .RemoveGraph(sep) — удаляет из текущего словаря полусиблинговых графов пару, ключ которой равен sep.

2.4. Выводы по главе

Сформирован набор определений, которые будут использоваться в рассуждениях в следующих главах, а также дано пояснение всем нестандартным командам и операциям, которые используются в листингах алгоритмов в 3-й главе.

3. Инкрементальные алгоритмы синтеза

3.1. Введение

В ходе исследования и решения поставленных задач были разработаны алгоритмы инкрементального синтеза третичной и четвертичной структур. В данной главе приводятся листинги с псевдокодом алгоритмов изменения третичной и четвертичной структур при добавлении новой вершины в АБС, а также при исключении вершины. Каждый алгоритм снабжен описанием и теоремой о его корректности.

3.2. Инкрементальное добавление вершины в третичную структуру

Заданы набор вершин V и построенная на их основе третичная структура АБС, построеная на множестве непустых сепараторов Sep.

При добавлении новой вершины u есть три варианта:

1) $\forall v \in V \;\; W_u \cap W_v = \emptyset$;

2) $\forall v \in V \;\; W_u \cap W_v \in \text{Sep}$ или $W_u \cap W_v = \emptyset$;

3) $\exists v \in V \;\; W_u \cap W_v \notin \text{Sep}$ и $W_u \cap W_v \neq \emptyset$.

В первом случае добавляемая вершина не порождает сепараторов, следовательно, третичная структура остается без изменений. Во втором случае сепараторы, получаемые при добавлении новой вершины, уже содержатся в множестве сепараторов Sep, на котором построена третичная структура, а значит изменений в ней не произойдет.

Рассмотрим подробнее третий случай, в котором появляются новые сепараторы от пересечений нагрузок добавляемой вершины с нагрузками из множества V, не принадлежащие множеству Sep.

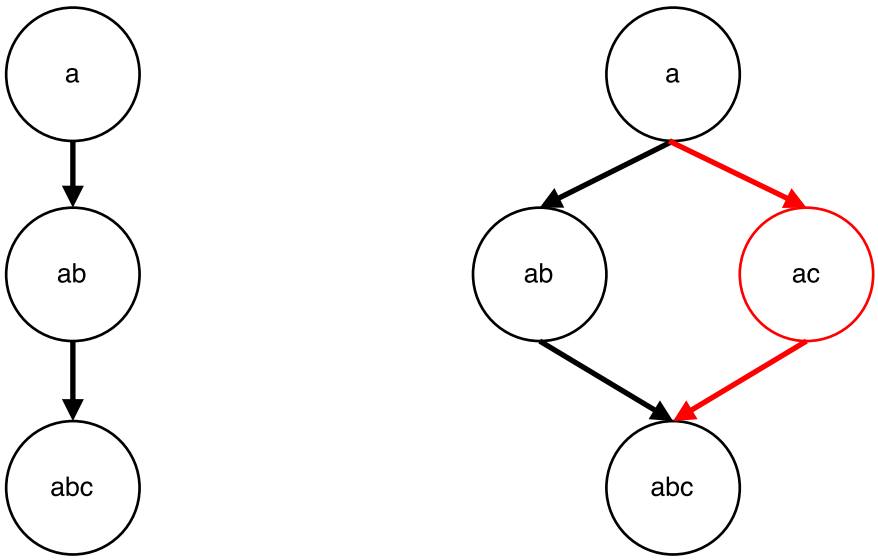

Рис. 1. Добавление нового сепаратора в третичную структуру.

На рис. 1 представлен пример добавления нового сепаратора в третичную структуру. Слева изображена третичная структура, построенная на множестве сепараторов $\{a, ab, abc\}$. Слева изображена та же третиччная структура, но с добавленным в нее сепаратором ac, причем новые ребра и вершина выделены красным цветом.

3.2.1. Описание алгоритма

На листинге 1 представлен псевдокод изменения третичной структуры при добавлении новой вершины в АБС. На вход подается построенная третичная структура $G = \langle S, E \rangle$, набор вершин первичной структуры V и новая вершина u. На выходе получаем измененую третичную структуру $G = \langle S', E' \rangle$. Для изменения третичной структуры сначала необходимо выделить новые сепараторы, порождаемые добавляемой вершиной, что осуществляется в строках 3–5, после чего получаем множество Sep_u, которое содержит все новые сепараторы.

Далее необходимо для каждого элемента s множества Sep_u (строки 6–31) построить множества родителей (строка 7) и сыновей (строка 8), содержащие элементы из множества S, которые должны быть соединены ребром с s в третичной структуре. Множество родителей для вершины s строится следующим образом: рассматриваем только вер-

Algorithm 1 Инкрементальный алгоритм изменения третичной структуры при добавлении новой вершины.

input: $G = \langle S, E \rangle$, V, u
output: $G = \langle S', E' \rangle$

```
1:  function AddSeparators
2:      Sep_u = ∅
3:      foreach (v in V)
4:          if ((W_v ∩ W_u ≠ ∅) and (W_v ∩ W_u not in V)) then
5:              Sep_u = Sep_u ∪ {(W_v ∩ W_u)}

6:      foreach (s in Sep_u)
7:          parents = ∅
8:          sons = ∅
9:          foreach (v in S)
10:             current = W_s ∩ W_v
11:             if (current ≠ ∅) then
12:                 f = true
13:                 if (current == W_v) then
14:                     foreach (p in parents)
15:                         if (W_p ∩ W_v == W_p) then
16:                             parents = parents \ {p}
17:                         else
18:                             if (W_p ∩ W_v == W_v) then
19:                                 f = false
20:                     if (f) then
21:                         parents = parents ∪ {W_v}
22:                 else
23:                     if (current == W_s) then
24:                         foreach (son in sons)
25:                             if (W_son ∩ W_s == W_s) then
26:                                 sons = sons \ {son}
27:                             else
28:                                 if (W_son ∩ W_s == W_son) then
29:                                     f = false
30:                         if (f) then
31:                             sons = sons ∪ {W_v}

32:         foreach (p in parents)
33:             foreach (son in sons)
34:                 if ((p, son) in E) then
35:                     E = E \ {(p, son)}
36:         S = S ∪ {s}

37:         foreach (p in parents)
38:             E = E ∪ {(p, s)}
39:         foreach (son in sons)
40:             E = E ∪ {(s, son)}
41:     return G
```

шины v из S, нагрузка которых содержится в нагрузке s (строка 13), затем удаляем из множества родителей все элементы, нагрузка которых содержится в W_v (строки 14–16), и если в множестве parents нет элемента, нагрузка которого содержит в себе W_v (строки 17–21), то добавляем элемент v в множество родителей.

Аналогично строится множество сыновей, отличие заключается в том, что оно будет содержать элементы, нагрузки которых содержат в себе W_s(строки 23–31). Таким образом мы построили множество parents, содержащее наибольшие по мощности вершины третичной структуры, нагрузки которых содержатся в W_s, и множество sons, содержащее наименьшие по мощности вершины третичной структуры, нагрузки которых содержат W_s.

Следующим шагом идет удаление существующих в третичной структуре ребер между элементами множества parents и sons, так как они будут соединены между собой через сепаратор s (строки 32–35). Далее добавляем новый сепаратор (строка 36) и новые ребра (строки 37–41) в множества вершин и ребер третичной структуры соответственно.

3.2.2. Обоснование корректности алгоритма

***Теорема* 3.1.** *Алгоритм, представленный на листинге 1 изменяет третичную структуру при добавлении новой вершины таким образом, что полученная структура является третичной для нового набора вершин.*

Доказательство. Чтобы граф $G = (S, E)$, построенный на множестве Sep являлся третичной структурой, необходимо выполнение следующих условий:

1. Множества Sep и S должны совпадать поэлементно.

2. Ребро между двумя вершинами u и v проведено тогда и только тогда, когда

$$W_u \cap W_v = W_u \text{ и } \nexists s \in S : \{W_s \cap W_u = W_s \text{ и } |W_s| > |W_u|\},$$

что соответствует правилу построения диаграмы Хассе.

На вход алгоритма подается граф, удовлетворяющий вышеперечисленным требованиям. Покажем, что структура, получаемая на выходе алгоритма также будет являться третичной.

Первое условие будет выполненно, так как все сепараторы, не содержащиеся в множестве вершин G (множество Sep_u), добавляются туда в строке 36. Таким образом, множество вершин в графе на выходе будет совпадать с множеством сепараторов.

Осталось проверить множество ребер: провести ребра к новым сепараторам и от них, а также удалить старые ребра, которые не будут удовлетворять правилу, описанному во втором условии.

Покажем, что второе условие также будет выполненно. Для каждого нового сепаратора s, добавляемого в третичную структуру, строятся множества возможных сыновей и родителей sons и parents соответственно. Причем, в sons попадают наименьшие по мощности сепараторы, пересечения которых с добавляемым содержат s. В parents, в свою очередь, попадают наибольшие по мощности сепараторы, пересечение которых с s входит в него. Вследствие чего ребра между новым сепаратором и вершинами, уже содержащимися в структуре будут проведены в соответствии с условием 2.

Но, если новый сепаратор стал соединять какие-либо вершины, то они должны были по построению третичной структуры быть соединены ребром, которое теперь нарушает второе условие. Поэтому, его необходимо удалить, что и происходит в строках 32–35.

Таким образом, алгоритм корректен и позволяет получить третичную структуру при добавлении новой вершины в первичную структуру АБС.

3.3. Декрементальное удаление вершины из третичной структуры

Заданы набор вершин V, множество непустых сепараторов Sep и третичная структура АБС, построенная на этом множестве. Рассмот-

рим случай удаления вершины из уже построенного графа.

При удалении вершины u есть два варианта.

1. Вершина не содержит в себе уникальных сепараторов,

$$\forall s \in \text{Sep} \;\; W_s \subset W_u \;\; \exists v, k \in V : v \neq u, k \neq u, W_v \cap W_k = W_s.$$

2. Вершина содержит в себе уникальные сепараторы,

$$\exists s \in \text{Sep} \;\; W_s \subset W_u \;\; \nexists v, k \in V : v \neq u, k \neq u, W_v \cap W_k = W_s.$$

В первом случае третичная структура не изменится, так как при удалении вершины множество сепараторов останется неизменным, ведь на каждый сепаратор, входящий в удаляемую вершину, существуют две отличные от удаляемой вершины, в которые он входит. Раз не изменилось множество сепараторов, то не изменится и третичная структура, построенная на этом множестве.

Во втором случае необходимо будет найти сепараторы, которые получены только пересечением нагрузок удаляемой вершины с какой-либо другой, и удалить их из третичной структуры с помощью функции RemoveSepFromThirdStructure, которая будет описана ниже.

3.3.1. Описание алгоритма

На листинге 2 представлен псевдокод изменения третичной структуры при удалении вершины из АБС. На вход подаются третичная структура $G = \langle S, E \rangle$, представленная в виде графа, набор вершин первичной структуры V, вершина u, которая будет удалена, и коллекция сепараторов Seps, представленная в виде словаря, в котором по ключу-сепаратору хранятся значения — сколько раз встречается данный сепаратор при пересечении каждой вершины с каждой.

Для изменения третичной структуры необходимо выделить все непустые сепараторы (строка 5), полученные при пересечении нагрузок данной вершины со всеми нагрузками остальных вершин (строки 3–4) и подсчитать сколько раз они встретились. Для этого будем использо-

вать структуру такого же типа, как и Seps (строки 6–9). Затем необходимо встречаемость каждого сепаратора sep из множества Seps, который принадлежит также и VertexSeps, уменьшить на соответствующее число из VertexSeps. Если после всех вышеперечисленных операций Seps[sep] будет равняться нулю, что равнозначно тому, что данный сепаратор образуется только пересечением нагрузки удаляемой вершины u с другими вершинами из множества S, то sep больше не будет присутствовать в множестве Seps после удаления вершины, следовательно он не будет присутствовать и в третичной структуре, построенной на множестве непустых сепараторов. Значит, сепаратор sep надлежит удалить из коллекции Seps (строка 14) и третичной структуры (строка 13).

Algorithm 2 Декрементальный алгоритм изменения третичной структуры при удалении вершины.

input: $G = \langle S, E \rangle$, V, u, Seps
output: $G = \langle S', E' \rangle$

```
 1: function RemoveSeparators
 2:     VertexSeps = ∅
 3:     foreach (v in V)
 4:         sep = W_v ∩ W_u
 5:         if (sep ≠ ∅)
 6:             if (VertexSeps.Contains(sep))
 7:                 VertexSeps[sep] = VertexSeps[sep] + 1
 8:             else
 9:                 VertexSeps.Add(sep, 1)
10:     foreach (sep in VertexSeps)
11:         Seps[sep] = Seps[sep] − sep.Value
12:         if (Seps[sep] == 0)
13:             RemoveSepFromThirdStructure(G, sep)
14:             Seps = Seps.Remove(sep)
15:     return G
```

3.3.2. Обоснование корректности алгоритма

Ниже представленно обоснование алгоритма с листинга 2. Описание функции RemoveSepFromThirdStructure и обоснование корректности будет представлено далее.

***Теорема* 3.2.** *Функция AddSeparators, представленная на листинге 2, при удалении вершины из первичной структуры АБС изменяет тре-*

тичную структуру так, что полученная структура является третичной для нового набора вершин.

Доказательство. Рассмотрим подробнее представленный алгоритм. Так как третичная структура построена на непустом множестве сепараторов, а рассматривается именно изменение третичной структуры, то нас интересует только изменение этого множества при удалении вершины. То есть, необходимо найти все сепараторы, которые исключатся из множества непустых сепараторов при удалении вершины, и удалить их из множества сепараторов и третичной структуры. Так как мы перебираем все вершины из множества V'', то в множестве VertexSeps получим все сепараторы, встречаемость которых уменьшится, и число, на которое она будет изменена. Затем в строках 10–11 пересчитаем встречаемость каждого сепаратора из множества сепараторов, и, если встречаемость станет равно 0, то удалим его из множества и из третичной структуры. Таким образом, данный алгоритм корректно изменяет множества сепараторов и третичную структуру при удалении вершины из первичной структуры. Доказательство корректности функции удаления сепаратора RemoveSepFromThirdStructure из третичной структуры предстваленно далее.

3.3.3. Описание алгоритма удаления сепаратора из третичной структуры

При удалении сепаратора sep из третичной структуры, представленной графом $G = (S, E)$ нужно, при необходимости, добавить новые ребра между родителями и сыновьями удаляемой вершины, которые были связаны через нее. Для этого переберем все ребра из множества E(строка 4), и для каждого ребра, которое содержит вершину sep добавим вторую вершину либо в множество сыновей данного сепаратора(строки 5–6), если ребро идет в вершину sep, либо в множество родителей(строки 7–8), если ребро проведено из sep. После этого удалим все ребра, соединяющие родителей с sep(строки 9–10) и sep с сыновьями(строки 11–12). Затем удалим сепаратор sep из множества вершин

графа(строка 13) и проверим, необходимо ли соединять родителей с сыновьями ребром, либо между ними уже есть путь в графе через другие вершины(строка 16). Если нет, то соединим их ребром(строка 17).

Algorithm 3 Алгоритм удаления сепаратора из третичной структуры.

input: $G = \langle S, E \rangle$, sep
output: $G = \langle S', E' \rangle$

```
 1: function RemoveSepFromThirdStructure
 2:     Sons = ∅
 3:     Parents = ∅
 4:     foreach (edge in E)
 5:         if (edge.Source == sep)
 6:             Sons = Sons ∪ edge.Target
 7:         if (edge.Target == sep)
 8:             Parents = Parents ∪ edge.Source
 9:     foreach (parent in Parents)
10:         E = E \ {(parent, sep)}
11:     foreach (son in Sons)
12:         E = E \ {(sep, son)}
13:     S = S \ sep
14:     foreach (parent in Parents)
15:         foreach (son in Sons)
16:             if (!PathExists(parent, son))
17:                 E = E ∪ {(parent, son)}
18:     return G
```

3.3.4. Обоснование корректности алгоритма

***Теорема* 3.3.** *Приведенный на листинге 3 алгоритм корректно удаляет из третичной структуры, представленной графом $G = (S, E)$ сепаратор sep таким образом, что на выходе получается новый граф, соответствующий третичной структуре, построенной на множестве непустых сепараторов, не содержащего sep.*

Доказательство. Чтобы граф $G = (S, E)$, построенный на множестве Sep, являлся третичной структурой, необходимо выполнение следующих условий.

1. Множества Sep и S должны совпадать поэлементно.

2. Ребро между двумя вершинами u и v проведено тогда и только тогда, когда

$W_u \cap W_v = W_u$ и $\nexists s \in S : \{W_s \cap W_u = W_s$ и $|W_s| > |W_u|\}$.

Так как граф уже построен и является третичной структурой, то первое условие выполненно, следовательно, при удалении сепаратора sep из множеств S и Sep, они не перестанут поэлементно совпадать. Осталось проверить выполнение второго условия после работы алгоритма.

Ребер с одной вершиной не появится, так как в строках 9–12 удаляются все ребра, содержащие вершину sep. Построенные в алгоритме множества Parents и Sons таковы, что

$$\forall p, s \; p \in \text{Parents}, s \in \text{Sons} : \; W_p \cap W_s = W_p.$$

Проверка, что $\nexists k \in V : \{W_k \cap W_s = W_k$ и $|W_p| < |W_k| < |W_s|\}$ проводится в строке 16 (любым алгоритмом поиска пути в графе, например поиском в глубину). Если путь существует, значит такая вершина k существует в графе, значит она уже соединена ребрами с вершинами p и s, следовательно ребро (p, s) добавлять в множество E не нужно. Если же путь не найден, то такой вершины s не существует, а значит вершины p и s должны быть соединены ребром, и это ребро добавляется в строке 17. Таким образом, после работы алгоритма полученный граф является третичной структурой для множества Sep $\setminus$ {sep}.

3.4. Инкрементальное добавление вершины в четвертичную структуру

Рассмотрим изменение четвертичной структуры при добавлении новой вершины в АБС. Так как четвертичная структура строится по третичной, первое, что надо сделать — это перестроить третичную структуру по алгоритму 1. Но, так как четвертичная структура представляет собой семейство полусиблинговых графов, построенных для каждого сепаратора из третичной структуры, то для ее изменения нам понадобятся следующие данные.

1. Все новые сепараторы, чтобы добавить потсроенные по множеству их сыновей полусбилинговые графы в четвертичную структуру.

2. Все вершины из третичной структуры, у которых изменилось множество сыновей, так как это отразится на соответствующем данной вершине полусиблинговом графе.

3. Все сепараторы, которые входят в новую вершину и уже содержатся в третичной структуре, так как могут появится новые ребра в уже построенном полусиблинговом графе, содержащем данные сепараторы.

Для этого внесем изменения в алгоритм, описанный в листинге 1. Измененный алгоритм представлен ниже на листинге 4.

Если же при добавлении новой вершины третичная структура не изменяется, то есть не появляется новых сепараторов, то в четвертичной структуре при этом изменения могут проявится только в виде новых ребер в отдельных полусиблинговых графах.

Будем возвращать кроме перестроенной третичной стркутуры три множества: Parents, содержащее все вершины из третичной структуры, у которых поменялось множество сыновей, NewSeps, содержащее все новые сепараторы, добавленные в третичную структуру в процессе работы алгоритма, и OldSeps, содержащее все сепараторы, которые входят в новую вершину и в третичную структуру, добавленные в третичную структуру в процессе работы алгоритма.

Таким образом мы получим все сепараторы, полусиблинговые графы которых изменятся при добавлении новой вершины в АБС, а также все сепараторы, между которыми могут появиться ребра. То есть, для изменение четвертичной структуры останется перестроить полусиблинговые графы для полученных множеств сепараторов Parents и NewSeps и проверить на добавление новых ребер сепараторы из OldSeps, у которых будет общий родитель в третичной структуре.

Algorithm 4 Модифицированный инкрементальный алгоритм изменения третичной структуры при добавлении новой вершины.

input: $G = \langle S, E \rangle$, V, u, Parents, NewSeps, OldSeps
output: $G = \langle S', E' \rangle$, Parents, NewSeps, OldSeps

```
1:  function AddSeparatorsFourth
2:      Sep_u = ∅
3:      foreach (v in V)
4:          if (W_v ∩ W_u ≠ ∅ and (W_v ∩ W_u not in V)) then
5:              Sep_u = Sep_u ∪ {(W_v ∩ W_u)}

6:      foreach (v in S)
7:          if (W_v ∩ W_u = W_v)
8:              OldSeps = OldSeps ∪ {v}
9:      foreach (s in Sep_u)
10:         parents = ∅
11:         sons = ∅
12:         foreach (v in S)
13:             current = W_s ∩ W_v
14:             if (current ≠ ∅) then
15:                 f = true
16:                 if (current == W_v) then
17:                     foreach (p in parents)
18:                         if (W_p ∩ W_v == W_p) then
19:                             parents = parents \ {p}
20:                         else
21:                             if (W_p ∩ W_v == W_v) then
22:                                 f = false
23:                     if (f) then
24:                         parents = parents ∪ {W_v}
25:                 else
26:                     if (current == W_s) then
27:                         foreach (son in sons)
28:                             if (W_son ∩ W_s == W_s) then
29:                                 sons = sons \ {son}
30:                             else
31:                                 if (W_son ∩ W_s == W_son) then
32:                                     f = false
33:                         if (f) then
34:                             sons = sons ∪ {W_v}
```

```
35:         foreach (p in parents)
36:             if (p not in Sep_u)
37:                 Parents = Parents ∪ {p}
38:             foreach (son in sons)
39:                 if ((p, son) in E) then
40:                     E = E \ {(p, son)}
41:         S = S ∪ {s}

42:         foreach (p in parents)
43:             E = E ∪ {(p, s)}
44:         foreach (son in sons)
45:             E = E ∪ {(s, son)}
46:     return G, Parents, Sep_u, OldSeps
```

3.4.1. Описание алгоритма

На листинге 5 представлен псевдокод изменения четвертичной структуры при добавлении новой вершины.

На вход алгоритму подается построенная четвертичная структура GraphsCollection, представленная в виде коллекции полусиблинговых графов, набор вершин V, необходимый для функции изменения третичной структуры, добавляемая вершина u и третичная структура, представленная графом G.

На выходе получаем измененную четвертичную структуру, представленную в виде коллекции полусиблинговых графов GraphCollection.

Сперва происходит вызов функции AddSeparatorsFourth (строка 4), которая вызывает алгоритм изменения третичной структуры при добавлении новой вершины и возвращает множество родителей Parents, у которых изменилось множество сыновей в процессе работы функции, и множество NewSeps, содержащее новые сепараторы, добавленные в третичную структуру, а также множество OldSeps, в котором содержатся сепараторы, входящие в новую вершину и в третичную структуру. После этого, необходимо перестроить полусиблинговые графы для каждого сепаратора, множество сыновей которого изменилось (строки 6–17), построить полусиблинговые графы для новых сепараторов (строки 18–26) и проверить добаление новых ребер между сепараторами из множества OldSeps (строки 27–32).

В процессе перебора элементов множества Parent создаем ссылку на

граф, соответствующий текущему сепаратору p (строка 7), получаем множество сыновей для данного сепаратора из измененной третичной структуры (строка 8). Далее удаляем из графа вершины и ребра, содержащие сепараторы, которые уже не являются сыновьями p (строки 9–11). После этого добавим в граф все новые сыновья (строки 13–14) и соединим ребрами добавленную вершину с остальными вершинами графа, если сужения на них максимального графа смежности пересекаются (строки 15–17).

Далее необходимо добавить в коллекцию полусиблинговых графов новые графы для каждого нового сепаратора из третичной структуры, то есть из множества NewSeps. Переберем все элементы множества NewSeps (строка 18), в процессе перебора для каждого текущего сепаратора s добавим новый граф в коллекцию (строка 19), создадим ссылку на этот граф (строка 18), получим множество сыновей для s (строка 21).

Переберем и добавим все элементы множества сыновей sons в множество вершин текущего графа currentGraph (строка 23). Останется только соединить ребрами те вершины графа, для которых сужения на данные сепараторы пересекаются, что и происходит в строках 24–26.

После этого переберем все элементы множества OldSeps (строки 27–28), получим родителя сепараторов из третичной структуры, либо null, если его нет (строка 29), и, если есть общий родитель, между текущими сепараторами нет ребра в полусбилинговом графе их родителя, и их сужения пересекаются (строки 30–32), то добавим новое ребро (строка 33) в полусиблинговый граф родителя.

Algorithm 5 Инкрементальный алгоритм изменения четвертичной структуры при добавлении новой вершины.

Input: GraphsCollection, V, u, $G = \langle S, E \rangle$,

Output: GraphsCollection

```
1: function AddSeparatorsToFourthStructure
2:     Parents = ∅
3:     NewSeps = ∅
4:     OldSeps = ∅
5:     AddSeparatorsFourth(G, V, u, Parents, NewSeps, OldSeps)
6:     foreach (p in Parents)
7:         currentGraph = GraphsCollection[p]
8:         sons = SonsFromThird(p, G)
9:         foreach (v in currentGraph.V)
10:            if (v not in sons)
11:                currentGraph.Remove(v)
12:        foreach (s in sons)
13:            if (s not in currentGraph.V)
14:                currentGraph.AddVertex(s)
15:                foreach (v in currentGraph.V)
16:                    if (ConstrictionIntersect(s, v) and v ≠ s)
17:                        currentGraph.AddEdge(s, v)
18:    foreach (s in NewSeps)
19:        GraphCollection.AddNewGraph(s, NewGraph)
20:        currentGraph = GraphsCollection[s]
21:        sons = SonsFromThird(s, G)
22:        foreach (son in sons)
23:            currentGraph.AddVertex(son)
24:            foreach (v in currentGraph.V)
25:                if (ConstrictionIntersect(v, s) and v ≠ s))
26:                    currentGraph.AddEdge(v, s)
27:    for (i = 0; i < OldSeps.Count − 1; i.Inc)
28:        for (j = i + 1; j < OldSeps.Count; j.Inc)
29:            parent = GetParent(OldSeps[i], OldSeps[j]])
30:                if (parent ≠ null and
31:                (not GraphCollection[parent].Edges.Contains({OldSeps[i], OldSeps[j]})
32:                and ConstrictionIntersect(OldSeps[i], OldSeps[j]))
33:                    GraphCollection[parent].AddEdge({OldSeps[i], OldSeps[j]})
34:    return GraphCollection
```

3.4.2. Обоснование корректности алгоритма

***Теорема* 3.4.** *Приведенный на листинге 5 алгоритм корректно изменяет четвертичную структуру при добавлении новой вершины так, что на выходе получается структура, соответствующая четвертичной для нового набора вершин.*

Доказательство. Так как четвертичная структура представляет собой семейство полусиблинговых графов, построенных для каждого сепаратора из третичной структуры, а при появлении новой вершины в третичной структуре могут появиться новые сепараторы, что повлечет за собой появление новый полусбилинговых графов, либо новых сепараторов не будет, но, в любом случае, могут изменится связи между старыми.

Таким образом, для изменения четвертичной структуры необходимо выполнить три действия.

1. Для каждого сепаратора из третичной структуры, у которого поменялось множество сыновей, перестроить соответствующий ему полусиблинговый граф из четвертичной структуры.
2. Для каждого нового сепаратора из третичной структуры, появившегося после добавления новой вершины, построить полусиблинговый граф и добавить его в четвертичную структуру.
3. Для всех старых сепараторов, которые уже входили в третичную структуру, проверить условия для проведения между ними ребра в полусиблинговом графе их родителя в случае отсутствия данного ребра.

Если все три действия будут выполнены, то получившееся в результате семейство графов будет являться четвертичной структурой для нового набора вершин.

Продемонстрируем, что после работы алгоритма с листинга 5 все вышеперечисленные действия будут выполнены.

Покажем, что первый пункт списка выполняется. После вызова функции AddSeparatorsFourth в переменной Parents будут лежать все сепа-

раторы из третичной структуры, множество сыновей у которых поменялось. Далее в строках 6–17 выделяется полусиблинговый граф, соответствующий текущему сепаратору p из Parents, в котором удаляются вершины и ребро, связанные с этими вершинами, которые уже не содержатся в текущем множестве сыновей для p из третичной структуры, а, следовательно, не должны содержаться в полусиблинговом графе для данного сепаратора (строки 9–11). Далее, для всех новых сепараторов s из множества сыновей sons сепаратора p, которые не содержатся в множестве вершин полусиблингового графа, по нему построенного, выполняются следующие действия: s добавляется в множество вершин (строка 14), затем проводятся ребра между s и остальными сепараторами из множества вершин графа в соответствии с определением полусиблингового графа (строки 15–17). Таким образом, для каждого сепаратора из третичной структуры, у которого изменилось множество сыновей, перестроился соответствующим образом полусиблинговый граф, то есть 1-й пункт выполнен.

Выполнению второго пункта списка соответствуют строки 18–26, в которых рассматриваются все новые сепараторы s (строка 18), создается и добавляется в четвертичную структуру новый полусиблинговый граф для s (строки 19–20), который заполняется вершинами и ребрами в соответствии с определением полусиблингового графа (строки 23–25), все сыновья добавляются во множество вершин (строка 23), и проводятся ребра между теми вершинами, сужения на которые максимального графа смежности пересекаются (строки 24–26). Таким образом, условие под номером 2 выполнено, а значит, получившееся семейство полусиблинговых графов является четвертичной структурой для нового набора вершин.

Третий пункт списка выполняется в строках 27–33. Происходит перебор всех элементов множества OldSeps (строки 27–29), проверяется наличие общего родителя (строка 29), и, если родитель существует, ребро между двумя сепараторами еще не проведено, но условия для проведения ребра из определения полусиблингового графа выполнены (строки 30–32), то добавляестя новое ребро (строка 33).

Таким образом, алгоритм корректен.

3.5. Декрементальное удаление вершины из четвертичной структуры

Рассмотрим изменение четвертичной структуры при удалении вершины из АБС. Ввиду того, что четвертичная структура строится по третичной, сперва необходимо соответствующим образом изменить третичную структуру с помощью алгоритмов, описанных выше.

Четвертичная структура представляет собой семейство полусиблинговых графов, построенных по сепараторам из третичной структуры, у которых есть сыновья. Таким образом, изменения в четвертичной структуре могут быть в трех случаях.

1. Если изменятся какие-либо полусиблинговые графы, что возможно, когда у соответствующих сепараторов в третичной структуре поменяется набор сыновей.

2. Изменится само семейство графов, то есть из него удалятся некоторые графы, что произойдет, когда удалятся соответствующие им сепараторы из третичной структуры.

3. Если у сепараторов в полусиблинговом перестанут пересекаться сужения после удаления вершины. Тогда ребро между ними необходимо будет удалить.

Таким образом, для изменения четвертичной структуры понадобится получить после работы алгоритма 2 дополнительные множества: набор сепараторов, которые были удалены из третичной структуры, набор вершин, у которых поменялось множество сыновей, и множество всех сепараторов из третичной структуры, которые входят в удаляемую вершину. Как и в случае с инкрементальным добавлением в четвертичную структуру, внесем необходимые изменения в алгоритм на листинге 2.

Будем возвращать множество удаляемых сепараторов DeletedSeps, сепараторы ChangedSeps, у которых изменилось множество сыновей, и множество OldSeps сепараторов, которые входят в удаляемую вершину.

Algorithm 6 Модифицированный декрементальный алгоритм изменения третичной структуры при удалении вершины.

input: $G = \langle S, E \rangle$, V, u, Seps, DeletedSeps, ChangedSeps , OldSeps
output: $G = \langle S', E' \rangle$, DeletedSeps, ChangedSeps

```
1:  function RemoveSeparatorsFourth
2:      VertexSeps = ∅
3:      foreach (v in V)
4:          sep = W_v ∩ W_u
5:          if (sep ≠ ∅)
6:              if (VertexSeps.Contains(sep))
7:                  VertexSeps[sep] = VertexSeps[sep] + 1
8:              else
9:                  VertexSeps.Add(sep, 1)
10:     foreach (sep in VertexSeps)
11:         Seps[sep] = Seps[sep] − sep.Value
12:         if (Seps[sep] == 0)
13:             ChangedSeps = ChangedSeps ∪ RemoveSepFromThirdStructure(G, sep)
14:             Seps = Seps.Remove(sep)
15:             DeletedSeps = DeletedSeps ∪ {sep}
16:     foreach (v in S)
17:         if (W_v ∩ W_u = W_v)
18:             OldSeps = OldSeps ∪ {v}
19:     return G, DeletedSeps, ChangedSeps, OldSeps
```

ChangedSeps будем подсчитывать в функции RemoveSepFromThirdStructure, для чего внесем необходимые изменения и в этот алгоритм.

При каждом удалении сепаратора из третичной структуры будем возвращать множество его родителей, тогда после работы алгоритма 6 в множестве ChangedSeps будут находиться все сепараторы, у которых был удален сын, что привело к изменению их множества сыновей. Так как в данном случае никаким другим способом множество сыновей у сепаратора поменяться не может, то в итоге получим необходимое множество.

3.5.1. Описание алгоритма

На листинге 8 представлен псевдокод алгоритма изменения четвертичной структуры при удалении вершины.

На вход алгоритму подаются четвертичная структура, представленная семейством графов GraphCollection, множество вершин V, удаля-

Algorithm 7 Модифицированный алгоритм удаления сепаратора из третичной структуры.

input: $G = \langle S, E \rangle$, sep
output: $G = \langle S', E' \rangle$, ChangedSeps

```
1:  function RemoveSepFromThirdStructure
2:      Sons = ∅
3:      Parents = ∅
4:      foreach (edge in E)
5:          if (edge.Source == sep)
6:              Sons = Sons ∪ edge.Target
7:          if (edge.Target == sep)
8:              Parents = Parents ∪ edge.Source
9:      foreach (parent in Parents)
10:         E = E \ {(parent, sep)}
11:     foreach (son in Sons)
12:         E = E \ {(sep, son)}
13:     S = S \ sep
14:     foreach (parent in Parents)
15:         foreach (son in Sons)
16:             if (!PathExists(parent, son))
17:                 E = E ∪ {(parent, son)}
18:     return G, Parents
```

емая вершина u, множество сепараторов Sep, третичная структура, представленная графом G. На выходе получаем измененную четвертичную структуру, соответствующую новому набору вершин.

Сперва необходимо изменить третичную структуру и получить множества сепараторов, полусиблинговые графы которых будут изменены (строка 4). Затем удалим из семейства графов все полусиблинговые графы (строки 5–6), сепараторы которых были удалены из третичной структуры в процессе работы функции RemoveSeparatorsFourth. После этого переберем все сепараторы ChangedSeps, множество сыновей которых изменилось, и перестроим соответствующие им полусиблинговые графы (строки 7–18). В процессе перебора для каждого сепаратора c из ChangedSeps удалим вершины из соответствующего ему полусиблингового графа currentGraph, которые уже не содержатся в множестве сыновей c (строки 9–12).

Осталось найти среди сыновей c сепараторы, которые еще не принадлежат множеству вершин currrentGraph.V (строки 13–14), добавить их в множества вершин (строка 15), после чего соединить добавленные

сепараторы с остальными вершинами в соответствии с определением полусиблингового графа (строки 16–18).

Algorithm 8 Декрементальный алгоритм изменения четвертичной структуры при удалении вершины.

Input: GraphsCollection, V, u, Seps, $G = \langle S, E \rangle$,
Output: GraphsCollection

```
1:  function RemoveSeparatorsToFourthStructure
2:      Parents = ∅
3:      DeletedSeps = ∅
4:      RemoveSeparatorsFourth(G, V, u, Seps, DeletedSeps, Parents)
5:      foreach (d in DeletedSeps)
6:          GraphCollection.RemoveGraph(d)
7:      foreach (c in ChangedSeps)
8:          currentGraph = GraphCollection[c]
9:          sons = SonsFromThird(c, G)
10:         foreach (v in currentGraph.V)
11:             if (v not in sons)
12:                 currentGraph.Remove(v)
13:         foreach (s in sons)
14:             if (s not in currentGraph.V)
15:                 currentGraph.AddVertex(s)
16:                 foreach (v in currentGraph.V)
17:                     if (ConstrictionIntersect(s, v) and v ≠ s)
18:                         currentGraph.AddEdge(s, v)
19:     return GraphCollection
```

3.5.2. Обоснование корректности алгоритма

***Теорема* 3.5.** *Приведенный на листинге 8 алгоритм корректно изменяет четвертичную структуру при удалении вершины так, что на выходе получается структура, соответствующая четвертичной для нового набора вершин.*

Доказательство. При удалении вершины четвертичная структура меняется тогда и только тогда, когда меняется третичная. Если меняется третичная, значит в нее добавляются новые сепараторы, либо, в нашем случае удаляются. Так как четвертичная структура представля-

ет собой семейство полусиблинговых графов, построенных для каждого сепаратора из третичной структуры, то нас интересуют два вида сепараторов из третичной структуры после ее изменения:

1) удаленные сепараторы;

2) сепараторы, у которых изменилось множество сыновей.

Если сепаратор не попадает ни под один из пунктов, то его полусиблинговый граф останется неизменным, значит, нужно только изменить для всех сепараторов, подходящих под один из пунктов, соответствующие им полусбилинговые графы, чтобы четвертичная структура соответствовала новому набору вершин.

Для удаленных сепараторов необходимо удалить из семейства их полусиблинговые графы, что и происходит в строках 5–6. Для сепараторов, у которых изменилось множество сыновей, необходимо перестроить соответствующий им полусбилинговый граф (строки 7–18), а именно, удалить все вершины, которые не входят в новое множество сыновей текущего сепаратора (строки 9–12) и добавить вершины, которые входят в множество сыновей, но еще не добавленны в множество вершин, а также соединить их ребрами с другими вершинами в по парвилам в соответствии с определением полусиблингового графа.

После всех вышеперечисленных действий, для всех вершин из третичной структуры, которые подходили под одно из двух условий, были изменены соответствующие им полусиблинговые графы в четвертичной структуре. Таким образом, полученная на выходе алгоритма структура являестя четвертичной для нового набора вершин.

3.6. Выводы по главе

В главе представлены представленны 4 алгоритма вместе с обоснованием их корректности, позволяющие перестраивать третичную и четвертичную структуру алгебраической байесовской сети при добавлении либо удалении вершины в первичную структуру. Таким образом, поставленные теоретические задачи были выполнены в полном объеме.

4. Руководство пользователя

4.1. Введение

Данная глава влкючает в себя руководство пользователя программным комплексом и описания разработанных классов и методов, с приведением реализации на языке C# некоторых из них.

4.2. Описание программной реализации

Для представления вершин в проекте были спроектированы и реализованы два интерфейса: IData, представляющий собой элемент вершины, хранящий тип string и ISeparable<IData>, служащий для хранения одной вершины в виде коллекции строк.

Во второй главе третичная структура определена как граф, построенный на множестве непустых сепараторов. В программной реализации к ней добавлен пустой сепаратор для удобства вычисления четвертичной структуры, а также для улучшения восприятия визуализации.

Далее будем для удобства опускать типы элементов, если они будут равны ISeparable<IData>.

Для хранения и работы с вершинами был разработан и реализован класс Vertices, содержащий в себе:

1) поле Alphabet, в котором находится алфавит текущего набора вершин;

2) поле Vertices, представляющее собой List, то есть список вершин;

3) поле Separators, представляющее собой словарь Dictionary<ISeparable<IData>>, int>, хранящий множество сепараторов текущего набора вершин, а так же частоту встречаемости каждого сепаратора в максимальном графе смежности;

4) метод добавления новой вершины AddVertex(vertex), добавляющий новую вершину vertex в список Vertices, а так же изменяющий соответственным образом алфавит и множество сепараторов;

5) метод RemoveVertex(vertex), схожий по функциональности с AddVertex, только удаляющий вершину из множества;

6) метод ConvertData(data), переводящий вершину data в бинарный вектор по текущему алфавиту;

7) метод Separator(vertex1, vertex2), вычисляющий сепаратор двух вершин с помощью метода ConvertData, что получается существенно быстрее, чем искать пересечение напрямую.

Код описанных выше методов и классов представлен на листинге 1.

Листинг 1. Класс Vertices

```
/// <summary>
/// Class to store fragments
/// </summary>
/// <typeparam name="T">type of atom of fragments</typeparam>
public class Vertices<T> where T : IData {
    /// <summary>
    /// Gets the alphabet.
    /// </summary>
    /// <value>The alphabet.</value>
    public List<T> Alphabet { get; }

    /// <summary>
    /// Gets the fragments.
    /// </summary>
    /// <value>The fragments.</value>
    public List<ISeparable<T>> Fragments { get; }

    /// <summary>
    /// Gets the separators.
    /// </summary>
    /// <value>The separators.</value>
    public Dictionary<ISeparable<T>, int> Separators { get; }

    /// <summary>
    /// Adds the vertex to alphabet.
    /// </summary>
    /// <param name="vertex">The vertex.</param>
    public void AddVertex(ISeparable<T> vertex) {
```

```
        if (Fragments.Contains(vertex)) return;
        var result = new VertexData<T>(vertex.ListOfData);
        Fragments.Add(result);
        AddVertexToAlphabet(result);
        AddSeparatorsFromVertex(vertex);
    }

    /// <summary>
    /// Removes the vertex.
    /// </summary>
    /// <param name="vertex">The vertex.</param>
    /// <returns><c>true</c> if XXXX, <c>false</c>
       otherwise.</returns>
    public bool RemoveVertex(ISeparable<T> vertex) {
        if (!Fragments.Contains(vertex))
            return false;
        Fragments.Remove(vertex);
        var f = false;
        var toRemove = new List<ISeparable<T>>();
        foreach (var data in vertex.ListOfData) {
            foreach (var separator in Separators.Where(separator
               => separator.Key
               .ListOfData.Contains(data))) {
                toRemove.Add(separator.Key);
                f = true;
                break;
            }
        }
        foreach (var separator in toRemove)
            Separators[separator]--;
        return f;
    }

    /// <summary>
    /// Converts the data.
    /// </summary>
    /// <param name="data">The data.</param>
    /// <returns>System.Byte[].</returns>
    private byte[] ConvertData(ISeparable<T> data) {
        var result = new byte[Alphabet.Count];
```

```
            var count = 0;
            foreach (var letter in Alphabet) {
                if (data.ListOfData
                .Contains(letter))
                    result[count] = 1;
                count++;
            }
            return result;
        }

        /// <summary>
        /// Gets the separator of two fargments
        /// </summary>
        /// <param name="vertex1">The vertex1.</param>
        /// <param name="vertex2">The vertex2.</param>
        /// <returns>ISeparable&lt;T&gt;.</returns>
        public ISeparable<T> Separator(ISeparable<T> vertex1,
           ISeparable<T> vertex2) {
            var v1 = ConvertData(vertex1);
            var v2 = ConvertData(vertex2);
            var length = Math.Min(v1.Length, v2.Length);
            if (length == 0) return null;
            var collection = new List<T>();
            for (var i = 0; i < length; i++) {
                if ((v1[i] & v2[i]) == 1)
                    collection.Add(Alphabet[i]);
            }
            return collection.Count != 0 ? new
               VertexData<T>(collection) : null;
        }
    }
```

Для хранения и работы с алгебраической байесовской сетью в проекте был разработан и реализовван класс AbnData, содержащий в себе:

1) поле Fragments для хранения ссылки на экземпляра класса Vertices;

2) поля _MaxAdjancyGraph, _secondaryStructure, _tertiaryStructure и _quaternaryStructure, хранящие соответственно максимальный граф смежности, вторичную, третичную и четвертичную структу-

ры;

3) приватные методы, реализующие изменение максимального графа смежности, третичной и четвертичной структур в случае удаления или добавления вершины с помощью алгоритмов, описанных в 3-й главе;

4) метод добавления новой вершины в алгебраическую байесовскую сеть AddVertex(vertex), добавляющий вершину vertex в поле Fragments, а так же перестраивающий все глобальные структуры;

5) метод удаления вершины из алгебраической байесовской сети RemoveVertex(vertex), удаляющий вершину vertex и перестраивающий все глобальные структуры с помощью алгоритмов, описанных в 3-й главе.

Код описанных выше методов и классов представлен на листинге 2

Листинг 2. Класс AbnData

```
/// <summary>
/// Class AbnData.
/// </summary>
 public class AbnData {
    /// <summary>
    /// Get fragments
    /// </summary>
    /// <value>The fragments.</value>
    public AbnDataClasses.Vertices<IData> Fragments { get; }

    /// <summary>
    /// The _max adjancy graph
    /// </summary>
    private BidirectionalGraph<ISeparable<IData>,
       AbnDataClasses.MyEdge<ISeparable<IData>>>
       _maxAdjancyGraph;

    /// <summary>
    /// The _secondary structure
    /// </summary>
    private BidirectionalGraph<ISeparable<IData>,
```

```
    IEdge<ISeparable<IData>>> _secondaryStructure;

/// <summary>
/// The _tertiary structure
/// </summary>
private BidirectionalGraph<ISeparable<IData>,
    IEdge<ISeparable<IData>>> _tertiaryStructure;

/// <summary>
/// The _quaternary structure
/// </summary>
private Dictionary<ISeparable<IData>,
    AbnDataClasses.HalfSiblingGraph
<ISeparable<IData>>> _quaternaryStructure;

/// <summary>
/// The _secondary structure
/// </summary>
/// <value>The secondary structure.</value>
public BidirectionalGraph<ISeparable<IData>,
    IEdge<ISeparable<IData>>> SecondaryStructure {
     get {
          return _secondaryStructure ?? GraphAlgorithms
          .GenerateMinAdjacencyGrahpGreedy(
          Fragments.Fragments);
     }
     set { _secondaryStructure = value; }
}

/// <summary>
/// The _max adjancy graph
/// </summary>
/// <value>The maximum adjancy graph.</value>
public BidirectionalGraph<ISeparable<IData>,
    AbnDataClasses.MyEdge<ISeparable<IData>>> MaxAdjancyGraph
    {
     get { return _maxAdjancyGraph ?? MakeMaxAdjancyGraph(); }
     set { _maxAdjancyGraph = value; }
}
```

```
    /// <summary>
    /// The _tertiary structure
    /// </summary>
    /// <value>The get tertiary structure.</value>
    public BidirectionalGraph<ISeparable<IData>,
       IEdge<ISeparable<IData>>> TertiaryStructure {
        get { return _tertiaryStructure ??
           MakeTertiaryStructure(); }
        set { _tertiaryStructure = value; }
    }

    /// <summary>
    /// The _quaternary structure
    /// </summary>
    /// <value>The Quaternary structure.</value>
    public Dictionary<ISeparable<IData>,
       AbnDataClasses.HalfSiblingGraph
    <ISeparable<IData>>> QuaternaryStructure {
        get { return _quaternaryStructure ??
           MakeQuaternaryStructure(); }
        set { _quaternaryStructure = value; }
    }
}
```

Помимо этого были спроектированы и реализованы несколько классов, облегчающих работу со всеми вышеперечисленными структурами:

1. HalfSiblingGraph<ISeparable<IData>>, содержащий в себе полусиблинговый граф и сепаратор, по которому он был построен.
2. Component<ISeparable<IData>>, используемый для хранения компоненты связности и сепаратора, по которому она выделалась из четвертичной структуры.
3. MyEdge<ISeparable<IData>>, содержащий в себе две вершины и их сепаратор.

Код описанных выше методов и классов представлен на листинге 3.

Листинг 3. Класс AbnDataClasses

```
 /// <summary>
/// Class AbnDataClasses.
/// </summary>
public class AbnDataClasses {
    /// <summary>
    /// Halfsibling graph which contains graph and separator
    /// </summary>
    /// <typeparam name="T">vertex type</typeparam>
    public class HalfSiblingGraph<T> where T : ISeparable<IData>
       {
        /// <summary>
        /// Gets the separator.
        /// </summary>
        /// <value>The separator.</value>
        public T Separator { get; }

        /// <summary>
        /// Gets or sets the graph.
        /// </summary>
        /// <value>The graph.</value>
        public BidirectionalGraph<T, IEdge<T>> Graph { get; set;
           }

        /// <summary>
        /// Initializes a new instance of the <see
           cref="HalfSiblingGraph{T}"/> class.
        /// </summary>
        /// <param name="separator">The separator.</param>
        /// <param name="graph">The graph.</param>
        public HalfSiblingGraph(T separator,
           BidirectionalGraph<T, IEdge<T>> graph) {
            Graph = graph;
            Separator = separator;
        }

        /// <summary>
        /// Returns a hash code for this instance.
        /// </summary>
        /// <returns>A hash code for this instance, suitable for
```

```
            use in hashing algorithms and data structures like a
            hash table.</returns>
        public override int GetHashCode() {
            return Separator.GetHashCode();
        }
    }
    /// <summary>
    /// Class Component.
    /// </summary>
    /// <typeparam name="T"></typeparam>
    public class Component<T> where T : ISeparable<IData> {
        /// <summary>
        /// Gets or sets the collection.
        /// </summary>
        /// <value>The collection.</value>
        public List<T> Collection { get; set; }

        /// <summary>
        /// Gets the separator.
        /// </summary>
        /// <value>The separator.</value>
        public T Separator { get; }

        /// <summary>
        /// Initializes a new instance of the <see
            cref="Component{T}"/> class.
        /// </summary>
        /// <param name="collection">The collection.</param>
        /// <param name="separator">The separator.</param>
        public Component(List<T> collection, T separator) {
            Separator = separator;
            Collection = collection;
        }
    }

    /// <summary>
    /// Class MyEdge.
    /// </summary>
    /// <typeparam name="T"></typeparam>
    /// <seealso cref="QuickGraph.IEdge{T}" />
```

```
public class MyEdge<T> : IEdge<T> where T :
    ISeparable<IData> {
    /// <summary>
    /// Initializes a new instance of the <see
        cref="MyEdge{T}"/> class.
    /// </summary>
    /// <param name="source">The source.</param>
    /// <param name="target">The target.</param>
    /// <param name="separator">The separator.</param>

    public MyEdge(T source, T target, T separator) {
        Source = source;
        Target = target;
        Separator = separator;
    }

    /// <summary>
    /// Gets the source vertex
    /// </summary>
    /// <value>The source.</value>
    /// <getter>
    ///   <ensures>Contract.Result&lt;TVertex&gt;() !=
        null</ensures>
    /// </getter>
    public T Source { get; }

    /// <summary>
    /// Gets the target vertex
    /// </summary>
    /// <value>The target.</value>
    /// <getter>
    ///   <ensures>Contract.Result&lt;TVertex&gt;() !=
        null</ensures>
    /// </getter>
    public T Target { get; }

    /// <summary>
    /// Gets the separator.
    /// </summary>
    /// <value>The separator.</value>
```

```
        public T Separator { get; }
    }
}
```

Методы добавления и удаления вершин в алгебраическую байесовскую сеть были неоднократно проверены с помощью UnitTests.

4.3. Примеры визуализации работы алгоритмов

Рассмотрим случайно сгенерированный набор из десяти вершин. Максимальный граф смежности для такого набора представлен на рис. 2.

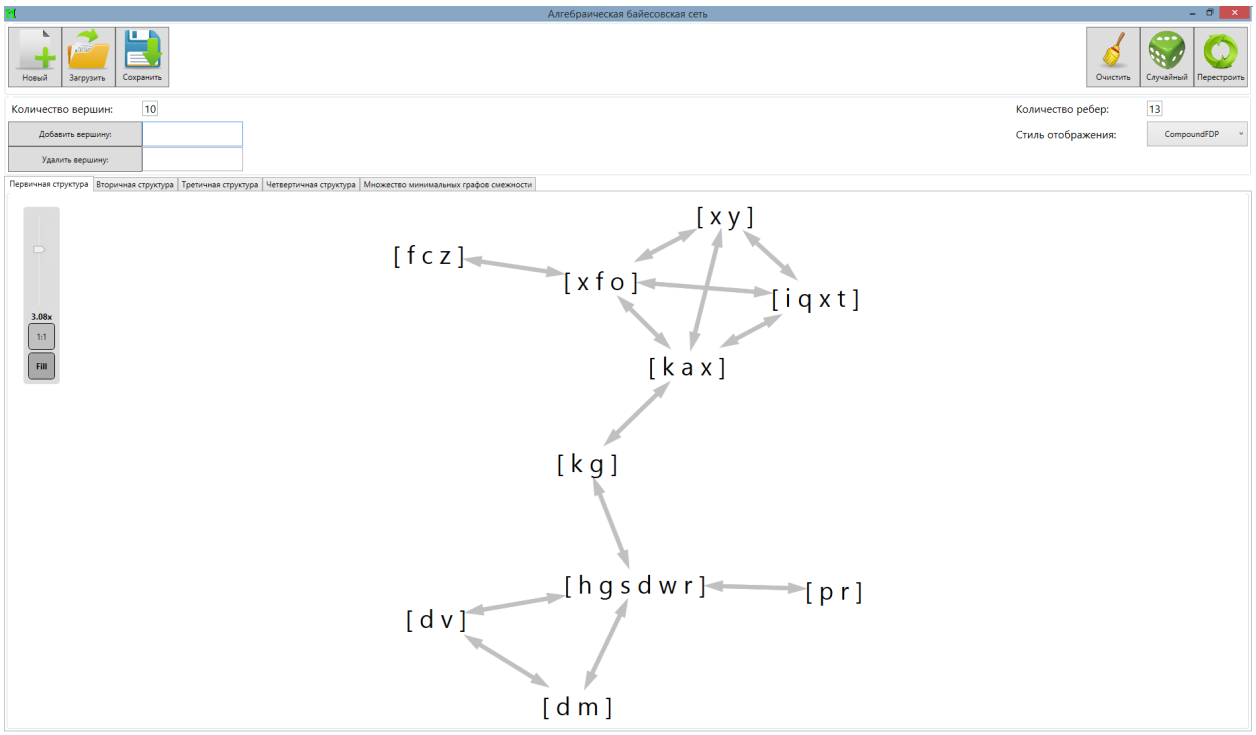

Рис. 2. Максимальный граф смежности.

Третичная структура для указанного выше набора вершин изображена на рис. 3, а четвертичная — на 4. В окне вывода для четвертичной структуры показан полусиблинговый граф, построенный на множестве сыновей для сепаратора из третичной структуры, выбранного из множества сепараторов, отображенных справа (в данном случае это граф для пустого сепаратора).

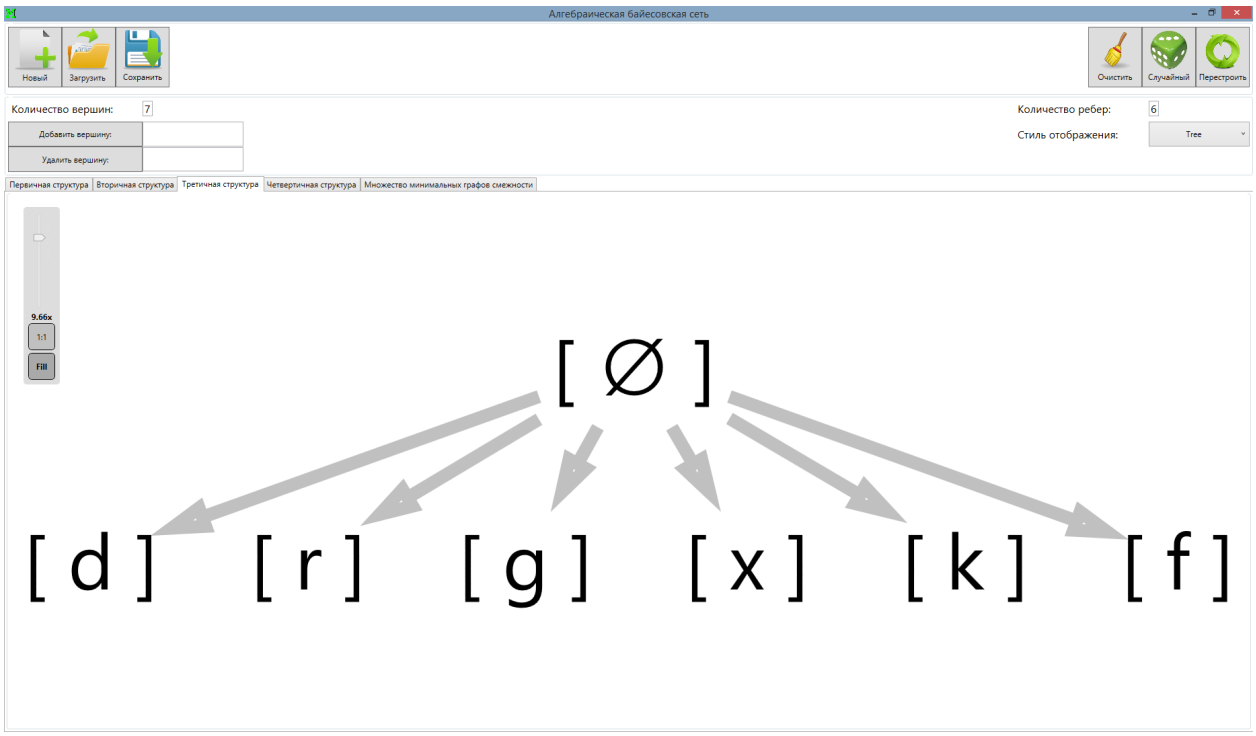

Рис. 3. Третичная структура.

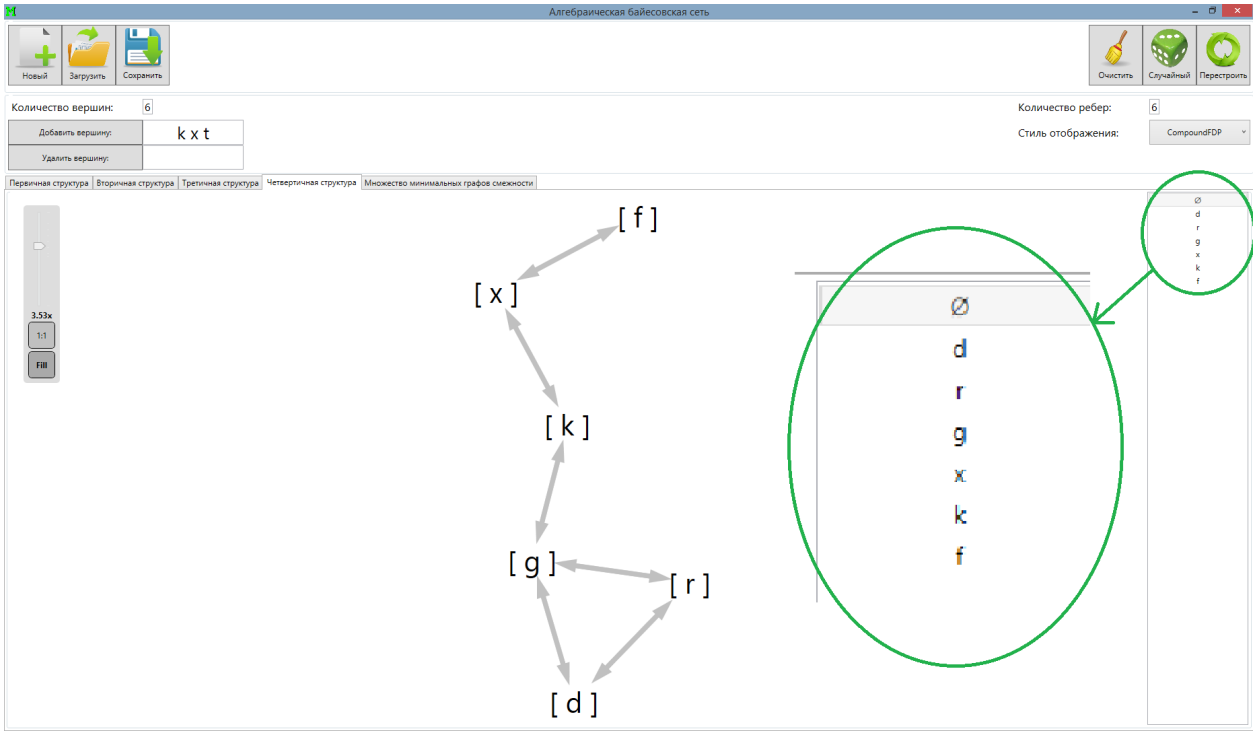

Рис. 4. Четвертичная структура.

Добавим в набор вершин новую — kxt. Тогда максимальный граф смежности изменится как показано на рис. 5. Заметно, что добавленная вершина имеет непустое пересечение (сепаратор) с еще пятью вершинами.

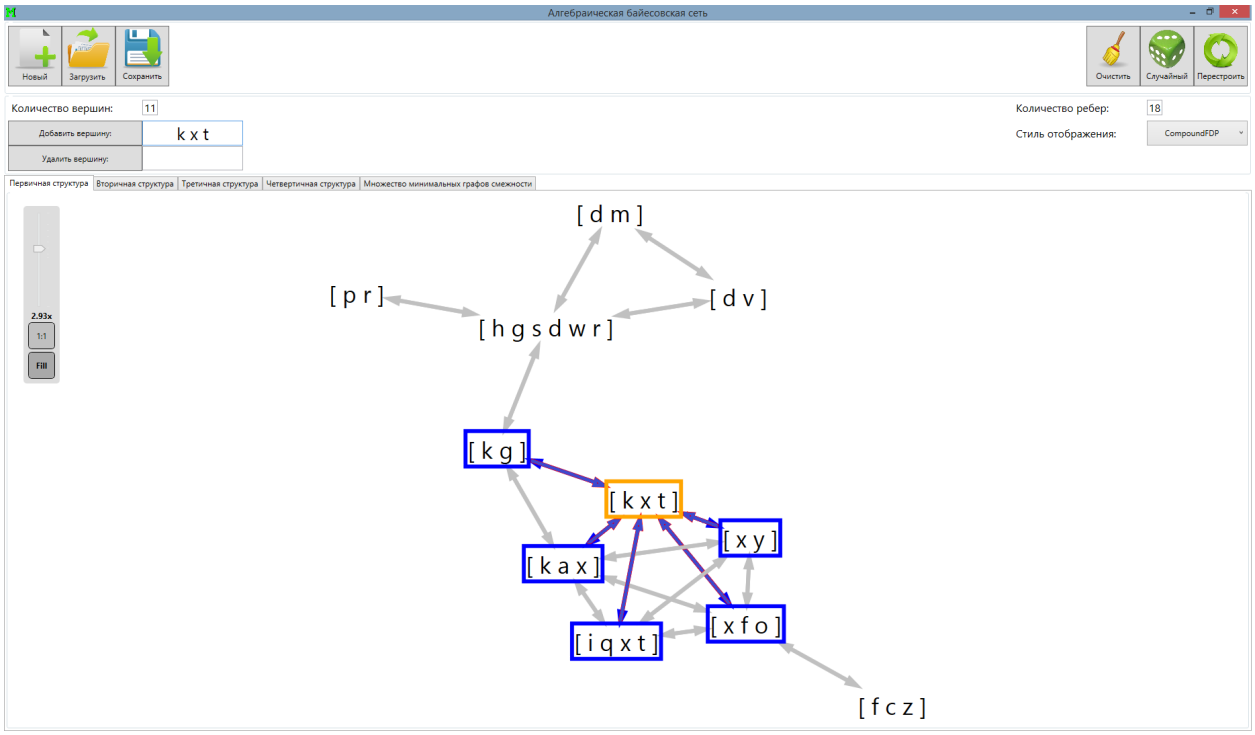

Рис. 5. Максимальный граф смежности с добавленной вершиной kxt.

После добавления вершины третичная структура изменилась: добавились сепараторы xt и xk, а также ребра, соединяющие их с родителями (рис. 6).

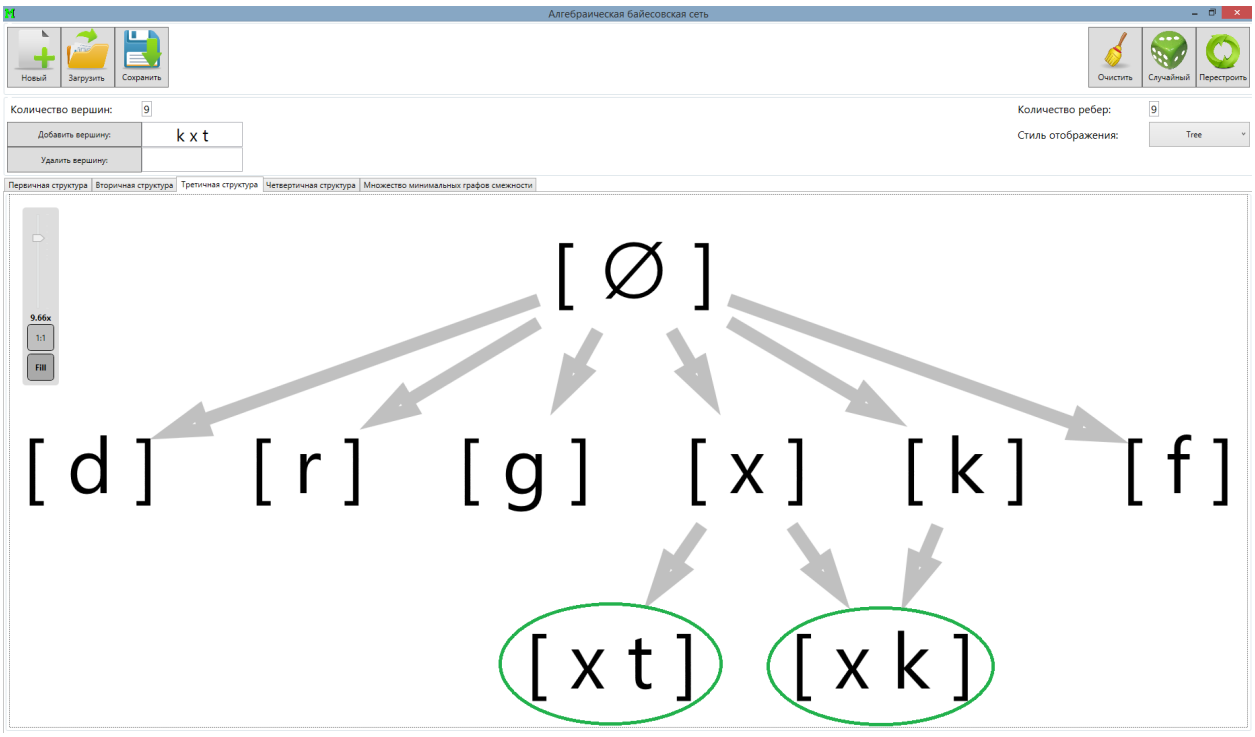

Рис. 6. Третичная структура с добавленной вершиной kxt.

Полусиблинговый граф для сепаратора x был пуст, так как у x не было сыновей в третичной структуре (рис. 7).

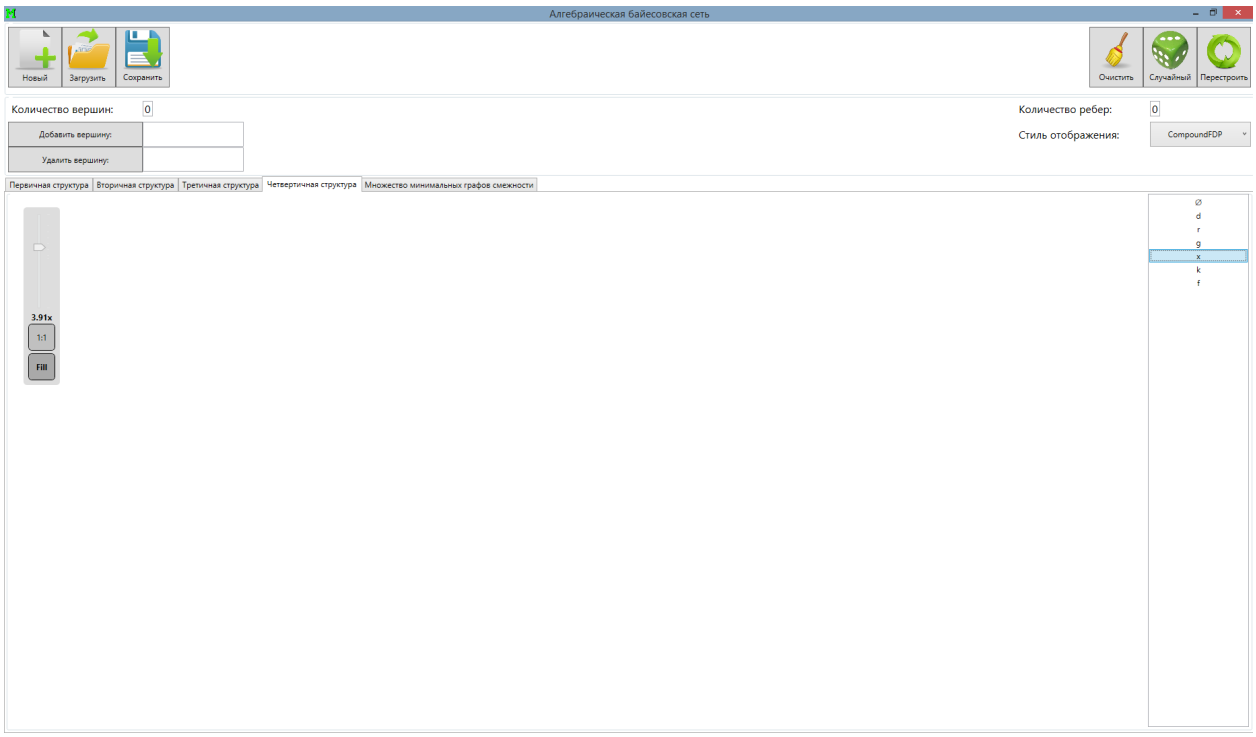

Рис. 7. Полусиблинговый граф для сепаратора x до добавления.

После добавления вершины, полусиблинговый граф для x изменился соотвутствующим образом (рис. 8). Также добавились в список новые сепараторы xk и xt.

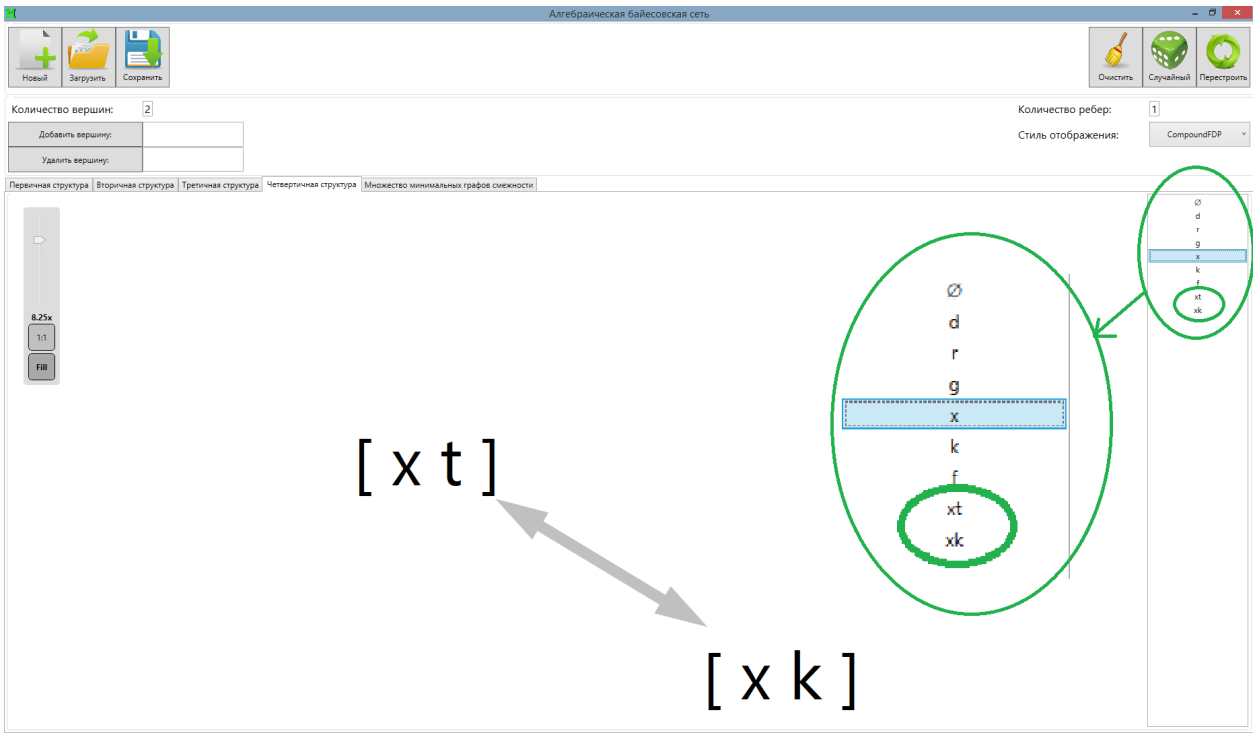

Рис. 8. Полусиблинговый граф для сепаратора x после добавления.

Удаление вершин из алгебраической байесовской сети происходит аналогичным образом, то есть все структуры перестраиваются для нового набора вершин.

4.4. Выводы по главе

Описана функциональность комплекса программ, представлено руководство пользователя, а также приведены фрагменты кода на языке C#.

Заключение

Поставленная цель работы — инкрементализация третичной и четвертичной структур алгебраических байесовских сетей — была достигнута. В ходе выполнения выпускной квалификационной работы бакалавра были разработаны 4 алгоритма — добавления и удаления вершин в третичную и четвертичную структуры АБС. Каждый из алгоритмов был протестирован и реализован в рамках библиотеки, интегрированной в комплекс программ по синтезу и анализу вторичной структуры алгебраических байесовских сетей. Кроме того, были доказаны 4 теоремы о корректности вышеупомянутых алгоритмов.

По теме работы была принята к публикации статья на конференцию IITI'16 (список SCOPUS), кроме того часть исследований была представлена на СПИСОК'16.

Список литературы

[1] Городецкий В.И. Алгебраические байесовские сети — новая парадигма экспертных систем // Юбилейный сборник трудов институтов Отделения информатики, вычислительной техники и автоматизации РАН. Т. 2. — 1993. — С. 120–141.

[2] Левенец Д.Г., Зотов М.А., Тулупьев А.Л. Инкрементальный алгоритм синтеза минимального графа смежности // Компьютерные инструменты в образовании. — 2015. — № 6. — С. 3–18.

[3] Николенко С.И., Тулупьев А.Л. Вероятностная семантика байесовских сетей в случае линейной цепочки фрагментов знаний // Труды СПИИРАН. — 2005. — № 2. — С. 53–57.

[4] Новиков Ф.А. Дикретная математика: учебник для вузов. 2-е изд. Стандарт третьего поколения. — СПб.: Питер, 2013. — С. 432.

[5] Опарин В.В., Тулупьев А.Л. Синтез графа смежности с минимальным числом ребер: формализация алгоритма и анализ его корректности // Труды СПИИРАН. — 2009. — № 11. — С. 142–157.

[6] Синтез вторичной структуры алгебраических байесовских сетей: инкрементальный алгоритм и статистическая оценка его сложности / Зотов М.А., Левенец Д.Г., Тулупьев А.Л., Золотин А.А. // Научно-технический вестник информационных технологий, механики и оптики. — 2016. — № 1. — С. 122–132.

[7] Сироткин А.В. Модели, алгоритмы и вычислительная сложность синтеза согласованных оценок истинности в алгебраических байесовских сетях // Информационно-измерительные и управляющие системы. — 2009. — № 11. — С. 32–37.

[8] Тулупьев А.Л., Николенко С.И., Сироткин А.В. Байесовские сети доверия: логико-вероятностный подход. — СПб.: Наука, 2006. — С. 607.

[9] Тулупьев А.Л., Сироткин А.В., Николенко С.И. Байесовские сети доверия: логико-вероятностный вывод в ациклических направленных графах. — СПб.: Издательство С.-Петербургского университета, 2009. — С. 400.

[10] Тулупьев А.Л. Алгебраические байесовские сети. Логико-вероятностный подход к моделированию баз знаний с неопределенностью. — СПб.: СПИИРАН, 2000. — С. 292.

[11] Тулупьев А.Л. Байесовские сети: логико-вероятностный вывод в циклах. — СПб.: Изд-во С.-Петербургского ун-та, 2008. — С. 140.

[12] Тулупьев А.Л. Задача локального автоматического обучения в алгебраических байесовских сетях: логико-вероятностный подход // Труды СПИИРАН. — 2008. — № 7. — С. 11–25.

[13] Тулупьев А.Л. Непротиворечивость оценок вероятностей в алгебраических байесовских сетях // Вестник СПбГУ. — 2009. — № 3. — С. 144–151.

[14] Тулупьев А.Л. Обработка дополнительной нечисловой информации в локальном обучении алгебраических байесовских сетей по выборкам с пропусками // Международная конференция по мягким вычислениям и измерениям. Сборник докладов. — 2009. — № 1. — С. 139–142.

[15] Тулупьев А.Л. Согласованность данных и оценка вероятности альтернатив в цикле стохастических предпочтений // Известия высших учебных заведений: Приборостроение. — 2009. — № 7. — С. 3–8.

[16] Тулупьев А.Л. Алгебраические байесовские сети: система операций глобального логико-вероятностного вывода // Информационно-измерительные и управляющие системы. — 2010. — № 11. — С. 65–72.

[17] Фильченков А.А., Тулупьев А.Л. Структурный анализ систем минимальных графов смежности // Труды СПИИРАН. — 2009. — № 11. — С. 104–127.

[18] Фильченков А.А., Тулупьев А.Л. Алгоритм выявления ацикличности первичной структуры алгебраической байесовской сети по ее четвертичной структуре // Труды СПИИРАН. — 2011. — № 19. — С. 128–145.

[19] Фильченков А.А., Тулупьев А.Л. Анализ циклов в минимальных графах смежности алгебраических байесовских сетей // Труды СПИИРАН. — 2011. — № 17. — С. 151–173.

[20] Фильченков А.А., Тулупьев А.Л. Третичная структура алгебраической байесовской сети // Труды СПИИРАН. — 2011. — № 18. — С. 164–187.

[21] Фильченков А.А., Фроленков К.В., Тулупьев А.Л. Устранение циклов во вторичной структуре алгебраической байесовской сети на основе анализа ее четвертичной структуры // Труды СПИИРАН. — 2012. — № 21. — С. 143–156.

[22] Фильченков А.А. Алгоритм построения множества минимальных графов смежности при помощи самоуправляемых клик // Труды СПИИРАН. — 2010. — № 12. — С. 119–133.

[23] Фильченков А.А. Алгоритм построения множества минимальных графов смежности при помощи самоуправляемых клик владений // Труды СПИИРАН. — 2010. — № 13. — С. 67–86.

[24] Фильченков А.А. Алгоритм построения множества минимальных графов смежности при помощи самоуправляемых клик-собственников // Труды СПИИРАН. — 2010. — № 14. — С. 150–169.

[25] Фильченков А.Л. Алгоритмы построения элементов третичной полиструктуры алгебраической байесовской сети // Труды СПИИРАН. — 2011. — № 18. — С. 237–266.

[26] Фильченков А.А. Синтез графов смежности в машинном обучении глобальных структур алгебраических байесовских сетей : Дисс... кандидата наук / А.А. Фильченков ; СПбГУ. — 2013.

[27] Microsoft. Introduction to the C# Language and the .NET Framework. — URL: https://msdn.microsoft.com/en-us/library/z1zx9t92.aspx (online; accessed: 22.03.2016).

[28] Microsoft. What's New in Visual Studio 2015. — URL: https://msdn.microsoft.com/en-us/library/bb386063.aspx (online; accessed: 22.03.2016).

[29] Neapolitan R.E. Learning Bayesian Networks. — Instock, 2004. — P. 674.